Local Pivotal Methods for Large Surveys

Jonathan J. Lisic * Nathan B. Cruze *

Abstract

The local pivotal method described provides a method to draw a spatially balanced sample for an arbitrary number of dimensions reducing the need for complex stratification for spatial surveys. Unfortunately, due to its quadratic run-time the local pivotal method has been restricted to populations with less than one million sampling units. In this paper, one alternative implementation and two approximations of this method are presented. The first uses a k-d tree data structure to implement the local pivotal method, which results in a sub-quadratic run-time. The second two methods relax the nearest-neighbor requirement in exchange for a further reduction in computational time through approximate nearest-neighbors. In this paper, an analysis of the effects of approximating spatial neighborhoods and a comparison of run-times between the proposed methods and existing implementations are provided for both simulated and real area frames.

Key Words: Nearest-Neighbor, Sampling, Spatial

1. Introduction

Tobler's First Law of Geography, Tobler (1970), states that "everything is related to everything else, but near things are more related than distant things." This is true for spatially close sampling units such as businesses, farms, and households; where neighboring sampling units are likely to have characteristics that exhibit spatial correlation. Because spatial correlation lowers the effective sample size when sampling units are close together (see Cressie, 1993), care should be taken to ensure that sampling units are well spread. Several proposed methods address this specific issue: the local pivotal method (LPM) (see Grafström et al., 2012), spatially correlated Poisson sampling (SCPS) (see Grafström, 2012), the generalized random-tessellation stratified method (GRTS) (see Stevens Jr and Olsen, 2004), and the draw-by-draw sampling excluding the selection of contiguous units (DDSESCU) (see Fattorini, 2006).

Unfortunately, none of these methods scale well to large populations. Both SCPS and LPM require nearest-neighbor searches, yielding quadratic computational complexity as seen in Grafström et al. (2012) and Grafström (2012) respectively. This particular problem is apparent in Grafström et al. (2014), where LPM is approximated to sample from a frame of 818,017 sampling units.

GRTS relies heavily on a data structure known as a quadtree. Quadtrees have excellent properties with respect to retrieving data, but can be computationally intensive to construct. This limitation can be seen in the R package *spsurvey* (see Kincaid and Olsen, 2015) where the number of levels of a tree are limited to 11, or a little over 4 million possible sampling units. DDSESCU, the drawn-by-drawn method, uses simulation to determine inclusion probabilities. This method is not usable for large population sizes due to the low probability of re-drawing the same unit.

This paper focuses on reducing the computational complexity of LPM, although a similar approach could be applied to SCPS. The approach taken here is through the use of k-dimensional trees (k-d trees) to accelerate nearest-neighbor searches. Section 2 of this paper provides background on LPM and k-d trees. Section 3 introduces an approximation

*United States Department of Agriculture, National Agricultural Statistics Service, 1400 Independence Avenue SW, Washington, DC 20250

to nearest-neighbor searching in k-d trees, which provides a greater decrease in computation complexity. Section 4 presents algorithms for both exact and approximate LPM using k-d trees. Simulated and empirical results for the accelerated LPM algorithm and its approximations are provided in Section 5. Finally, a discussion of the results and possible future work are provided in Section 6.

2. Background

The method presented in this paper is built on two key ideas, the local pivotal method and k-dimensional trees. LPM provides a method to create well spread or spatially balanced samples (Figure 1). This method creates spatial balanced samples by locally aggregating inclusion probabilities from neighboring sampling units, lowering the probability that adjacent sampling units are sampled. k-d trees assist this method by providing a computationally efficient means to identify neighboring sampling units.

Before introducing the LPM, it is important to first define spatial balance. In this paper spatial balance follows the definition provided by Stevens Jr and Olsen (2004),

$$B = \sum_{i \in \mathcal{S}} \left(-1 + \sum_{j \in \mathcal{N}_i} \pi_j \right)^2 \tag{1}$$

where

$\mathcal{S}$ = sample of size n from a population $\mathcal{U}$ of size N;
$\mathcal{N}_i$ = spatial neighborhood (Voronoi tessellation about sampled point i);
π_i = $\Pr(i \in \mathcal{S})$, the first order inclusion probability in sample $\mathcal{S} \subset \mathcal{U}$.

Necessarily, $\sum_{i \in \mathcal{U}} \pi_i = n$ and $1 \geq \pi_i > 0 \; \forall i \in \mathcal{U}$.

B is bounded on the interval $[0, n(n-1))$. The lower bound occurs when the sample is perfectly balanced; the sum of first order inclusion probabilities over each tessellate is one. The upper bound occurs under two conditions. First, the sampling weights for $N - n$ members of the frame have a small inclusion probability π^*, identified as $\mathcal{A}$, and n members of the frame have inclusion probabilities close to one, identified as $\mathcal{A}'$. Furthermore, the sets $\mathcal{A}$ and $\mathcal{A}'$ are mutually disjoint; $\mathcal{A} \cup \mathcal{A}' = \mathcal{U}$ and $\mathcal{A} \cap \mathcal{A}' = \emptyset$. Second, the members of $\mathcal{A}$ are sufficiently isolated in the spatial support from $\mathcal{A}'$ such that there exists a sample with a tessellate containing all n members of $\mathcal{A}'$. If this specific sample is drawn, then a single tessellate has a sum of inclusion probabilities over $\mathcal{N}_i$ close to n, where $i \in \mathcal{A}'$. All other tessellates have sums over $\mathcal{N}_i$ close to zero, $i \in \mathcal{A}$. In the limit, as π^* goes to zero, then B goes to $n - 1 + (n-1)^2 = n(n-1)$. This limit can be verified through Lagrange optimization over π_i, where $-B$ is minimized with the constraint $\sum_{i \in \mathcal{U}} \pi_i - n = 0$.

An intuitive case for this measure of balance occurs when the first order inclusion probabilities are equal and the sampling units in the population are evenly spread across the spatial domain. In this case, if each tessellate has the same area, then B is zero; a consequence of sampled points being well spread. Although it should be noted that B can be zero when the sampled points are adjacent to each other, as in the case of the sampled points forming a circle at the center of a disc-shaped spatial region. However, in practice this later case is rare.

k-d trees, like the previously mentioned tessellates, partition the spatial domain into a set of mutually disjoint and connected regions. In the case of k-d trees the partitioning is done through recursively bisecting the spatial domain by a univariate statistic such as a median. The univariate statistic is applied to each dimension of the k-d tree either by iterating over the index of the dimensions or through a heuristic, such as picking the dimension with the largest marginal variance. This partitioning algorithm allows for the construction of a

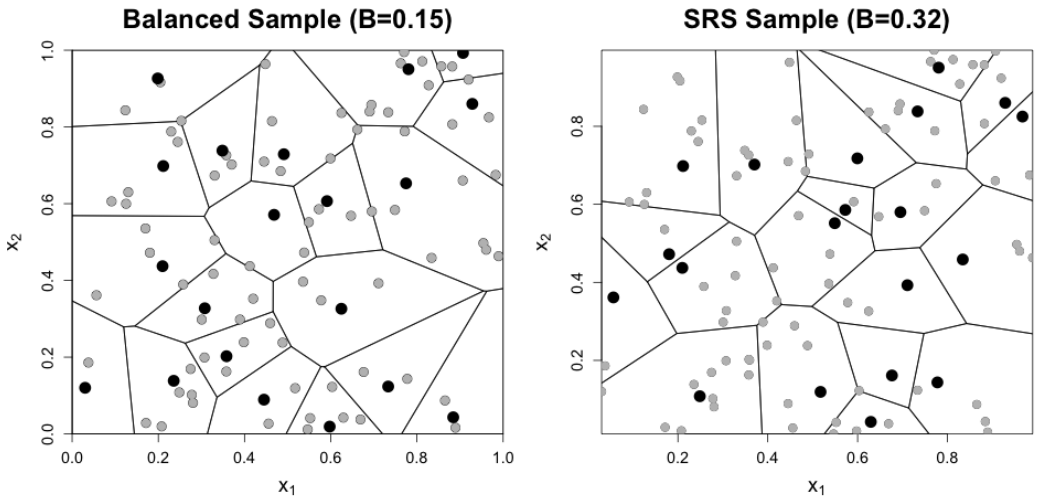

Figure 1: Left, a balanced sample with few clustered sampled units; right, a simple random sample (SRS) with some clustered sample units. The 20 sampled units in each example are in black, unsampled members of the frame are in grey. The frame is generated from an i.i.d. sample from $X_1 \sim \text{Uniform}(0,1)$, $X_2 \sim \text{Uniform}(0,1)$ of size 100.

binary tree that efficiently finds nearest-neighbors in a k-dimensional spatial domain. The details of both of these algorithms follow with emphasis on computational complexity.

2.1 Local Pivotal Method

The pivotal method is first developed by Deville and Tille (1998) as a method to efficiently calculate unequal probability designs through splitting first order inclusion probabilities. An adaption of the pivotal method to provide spatial balance can be found in Grafström et al. (2012). This paper provides two algorithms LPM1 and LPM2. In this paper, only the simpler and faster LPM2 will be considered.

The algorithm for LPM2 is straight-forward to implement (see Algorithm 1), and consists of two key components. The first component is the vector valued update function g. This function is defined as

$$g(i,j) = \begin{cases} (0, \pi_i + \pi_j) & \text{with probability} \frac{\pi_j}{\pi_i + \pi_j} \\ (\pi_i + \pi_j, 0) & \text{else} \end{cases} \tag{2}$$

when $\pi_i + \pi_j > 1$, otherwise

$$g(i,j) = \begin{cases} (1, \pi_i + \pi_j - 1) & \text{with probability} \frac{1-\pi_j}{2-\pi_i-\pi_j} \\ (\pi_i + \pi_j - 1, 1) & \text{else.} \end{cases} \tag{3}$$

The second key component, and slowest part of this algorithm occurs in line 3 of Algorithm 1, finding the nearest-neighbor of sampling unit i. The original implementation of this algorithm used an exhaustive search to accomplish this task. An exhaustive search requires calculating close to $N-k$ distance calculations at iteration $k \in \{1, \cdots, N-n-1\}$. This leads to the average computational of $\mathcal{O}\left(N^2\right)$ for this algorithm.

Algorithm 1 LPM2 Algorithm.

while length $(\mathcal{U}^*) > 0$ **do**
    Randomly select sampling unit $i \in \mathcal{U}^*$ with uniform probability.
    Set j to the nearest-neighbor of i in $\mathcal{U}^*$.
    Set $(\pi_i, \pi_j) := g(\pi_i, \pi_j)$.
    Set $\mathcal{U}^* := \mathcal{U}^* \backslash \{k \in \{i, j\} : \pi_k \in \{0, 1\}\}$.
end while

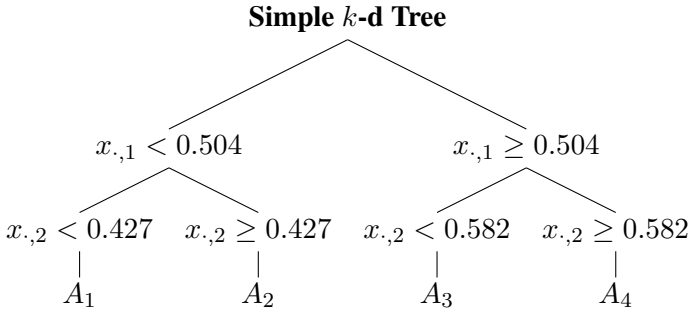

Figure 2: k-d tree node structure with depth = 2 applied to the population in Figure 1.

2.2 k-Dimensional Trees

Nearest-neighbors algorithms are integral in common statistical techniques such as k-nearest-neighbor clustering and calculation of high dimensional kernel density estimates. In both of these cases, k-d trees and variants provide a data structure that allows for nearest-neighbor queries with average computational complexity of $\mathcal{O}(\log(n))$ (see Muja and Lowe, 2014 and Lang et al., 2005). This search time is lower than the linear search used in the original LPM2 algorithm, but comes at the expense of building the k-d tree. k-d tree construction has $\mathcal{O}(n \log(n))$ computational complexity. Since computational complexity is defined up to a constant, the average computational complexity for tree construction and n queries against the tree is also $\mathcal{O}(n \log(n))$. Therefore, even with tree construction the average computational complexity of the n k-d tree searches is much lower than the quadratic computational complexity of the linear search.

k-d trees partition a set of k-dimensional points into a set of mutually exclusive subsets. This partitioning is done by splitting the original k-dimensional space along a single dimension at the median or other statistic calculated from the points in the space. Subsequent applications of this partitioning are performed individually on the resulting partitions, choosing a different dimension at each iteration. The choice of dimension can be made by simply iterating over each dimension repeatedly, or by selecting the dimension based on a heuristic such as the largest marginal variance. Tree construction terminates when the number of points in each subspace is less than or equal to a fixed number of nodes, where the number is set a priori. A graphical view of a k-d tree applied to data in Figure 1 with leaf node size m equal to 25 is provided in Figure 2 with the partitioning of the spatial support depicted in Figure 3. Further partitioning by the algorithm can be seen in Figure 4 with m equal to 13 and 7.

An algorithm to build a k-d tree-based on iterating through dimensions is detailed as Algorithm 2. In this algorithm the set of k-dimensional points $\mathcal{X}$ are indexed by the index $\mathcal{I}$. During each split, the sets and indexes are split between the child nodes. Note that, the algorithm provided is known as a leaf based k-d tree since all points are stored in the leafs as opposed to within the nodes of the tree, non-leaf based approaches are not pursued in this research.

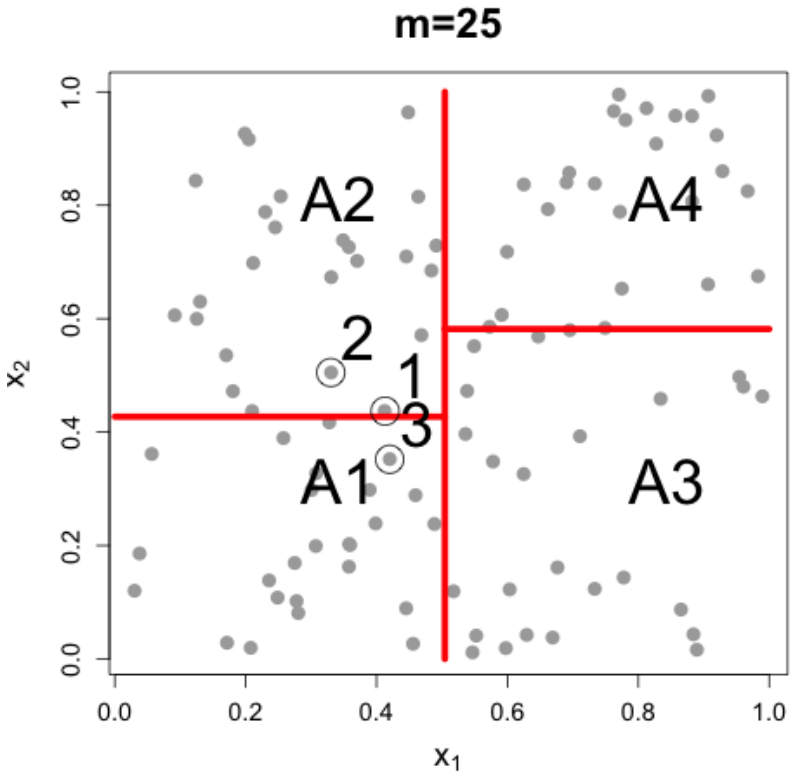

Figure 3: Labeled partitions of a k-d tree with max leaf node size of $m = 25$ from Figure 2. A query point (labeled 1) and its nearest-neighbors in A_1 (labeled point 3) and A_2 (labeled point 2) are circled.

Once a tree is constructed it can be queried to find neighbors. In the case of LPM and related methods, the query of interest is to find the nearest-neighbor of a point in the k-d tree. Since the point of interest is within the k-d tree, care needs to be taken to avoid returning the point queried. This can be avoided by performing an index check before comparing distances; if the index being queried matches the point to be compared, then this point is skipped. An algorithm that will return the nearest-neighbor of a point in the k-d tree is provided as Algorithm 3. To simplify this algorithm, no attempt is made to handle ties between distances. Such ties are common if applied to pixel data, and can be handled through reservoir sampling (see Sunter, 1977).

The search algorithm goes through three phases to find the nearest-neighbor for a query point in the tree. The first phase is a depth first search to find a partition of space that contains points close to the query point. Once the first leaf node is encountered phase two begins. In phase two, each point in the partition associated with the leaf node is compared to the query point. Care is taken to ensure comparisons are not done between the query point and itself. If a point is found closer to the point being queried than any prior point evaluated, it is set as the current nearest-neighbor and the distance between this point and the queried point is retained. After all points in the partition of point have been compared, the tree will backtrack to other nodes in the tree. However, in this third phase, only nodes that could feasibly lead to a point sufficiently close to the query point are considered. This feasibility is determined through the univariate median used for splitting in the k-d tree. If the distance between the query point and the univariate median for a given node is greater than the current distance, this node and all of its children will not be visited. This greatly

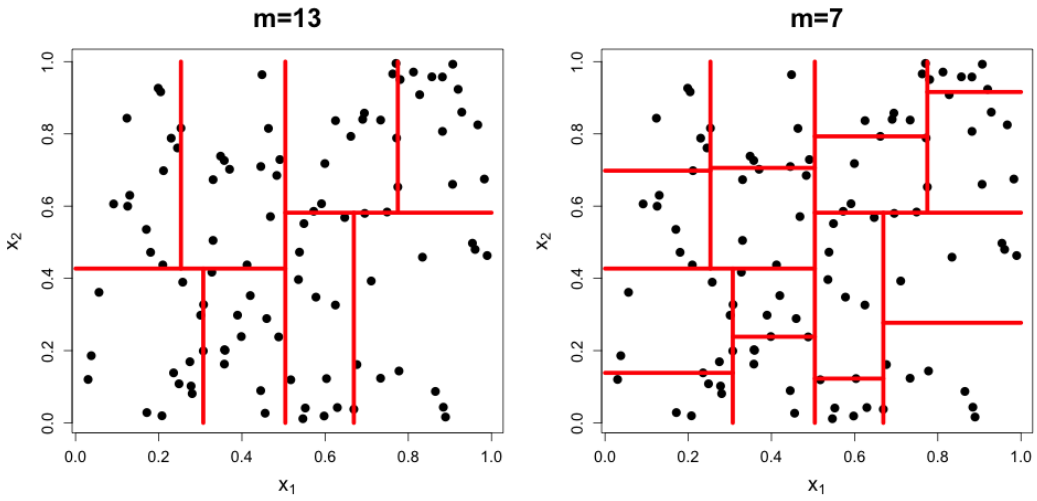

Figure 4: Labeled partitions of a k-d trees with max leaf node size of $m = 13$ (left) and $m = 7$ (right) from Figure 3.

Algorithm 2 Algorithm to build a k-d tree for a k-dimensional data set.

initialize:
    Index the points in the set $\mathcal{X}$ with $\mathcal{I}$
    $i \leftarrow 1$
procedure KDTREE($\mathcal{I}$,i)
    Create new node A, with attributes left, right, median, and data.
    if length $(\mathcal{I}) > m$ **then**
        Split $\mathcal{I}$ into $\mathcal{I}_l$ and $\mathcal{I}_r$ by the median of dimension i modulo k plus one.
        Set A's median to the split point.
        Set A's left child to KDTREE($\mathcal{I}_l$,$i + 1$).
        Set A's right child to KDTREE($\mathcal{I}_r$,$i + 1$).
    else
        Set A's data to $\mathcal{I}$.
    end if
    return A
end procedure

reduces the number of nodes to explore. These three phases are revisited until all viable nodes are traversed.

As an illustrative example, a query to find the nearest-neighbor of point $v = (0.413, 0.437)$ in a tree with leaf nodes of size 25 is provided. This point is identified in Figure 3 as point 1, and the tree in Figure 5 identifies the nodes visited in the query of this tree in bold. In the first phase the left node is selected since the first element of v, 0.413, is less than the univariate median in the first dimension of X; the right node is selected next since the second element of v is greater than or equal to 0.427. Phase two begins with the exhaustive nearest-neighbor search of all points in the subset of the partition labeled A_2; this search concludes with identification of the nearest-neighbor, point $(0.330, 0.505)$ with distance 0.011 (identified as point 2). In phase three, the distance between the second element of v and the univariate median of the second dimension is calculated as 0.0001. Since this distance is less than the current distance of 0.011, partition A_1 is also explored resulting in a nearest-neighbor of $(0.420, 0.352)$ with distance 0.007 (identified as point 3). Finally

Algorithm 3 Algorithm to query a k-d tree for the nearest-neighbor using the L^2 norm.

initialize:
  Index the points in the set $\mathcal{X}$ with $\mathcal{I}$
  Query for index $a \in \mathcal{I}$
  $y \leftarrow x_a \in \mathcal{X}$
  $dist \leftarrow \infty$
procedure SEARCH(A,i,(neighbor, dist))
  if A's median is defined. **then**
    Set j to i modulo k.
    $q \leftarrow A's$ median
    if $y_j \leq q$ **then**
      (neighbor, dist) $\leftarrow$ SEARCH(A_l,$i+1$,(neighbor, dist)).
      if $(y_j - q)^2 <$ dist **then**
        (neighbor, dist) $\leftarrow$ SEARCH(A_r,$i+1$,(neighbor, dist)).
      end if
    else
      (neighbor, dist) $\leftarrow$ SEARCH(A_r,$i+1$,(neighbor, dist)).
      if $(y_j - q)^2 <$ dist **then**
        (neighbor, dist) $\leftarrow$ SEARCH(A_l,$i+1$,(neighbor, dist)).
      end if
    end if
  else
    for b in A's data **do**
      $z \leftarrow x_b$
      if $a \neq b$ **then**
        if $||y - z||_2^2 <$ dist **then**
          dist $\leftarrow ||y - z||_2^2$
          neighbor $\leftarrow b$
        end if
      end if
    end for
  end if
  return (neighbor, dist)
end procedure

we enter phase 3 again and the distance is calculated between the first element of v and the univariate median of the first dimension, here the distance is 0.008 exceeding our prior minimum distance of 0.007. Since the minimum distance is exceeded there are no checks done on the right hand side of the tree. At this point the query terminates because there are no further nodes to check. The point $(0.420, 0.352)$ is returned as the nearest-neighbor.

3. Approximate Nearest-Neighbors

The nearest-neighbor literature also considers approximate nearest-neighbors (ANN) (see Arya et al., 1998) as a means to accelerate searches by sacrificing the requirement for closest neighbors. Instead of closest neighbors, the requirement is replaced with the closest neighbors found in a fixed amount of time or within a minimum distance. This form of approximation can be useful in application to LPM2, by sacrificing balance for speed. In this research, two methods of ANN are considered. The first is time-based. In this method

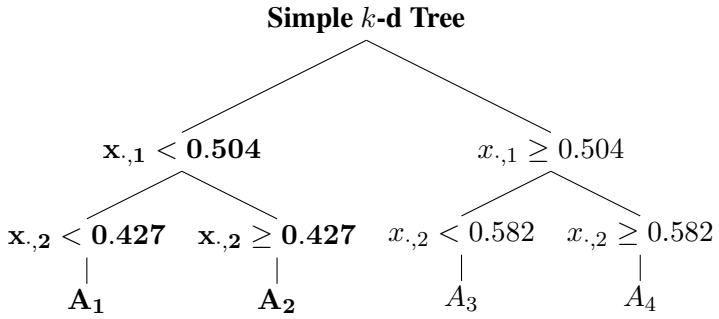

Figure 5: A search for the nearest-neighbor of point $(0.413, 0.437)$ (point 1. in Figure 3).

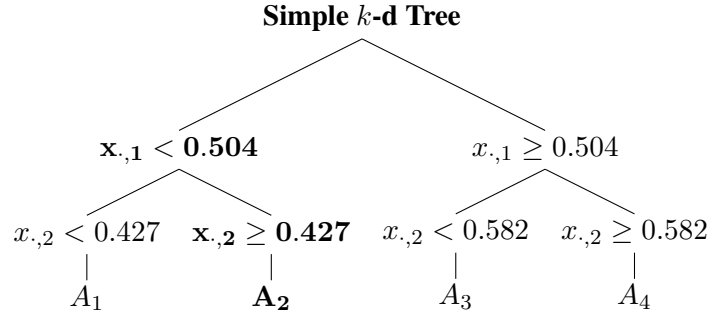

Figure 6: A search for the nearest-neighbor of point $(0.413, 0.437)$ (point 1. in Figure 3), using the time-based ANN (leaf check = 1).

the amount of time to find the closest neighbor is fixed, where time is specified as the number of partitions (leaf nodes) to check. The second is distance based, where the first neighbor found that is sufficiently close to the sampled point will be taken as an ANN.

Time-based ANN methods restrict searching for neighbors to a subset of the spatial domain. This improves the worst case performance of k-d trees (fixing number of leaves to check), avoiding traversals over large portions of the tree. The path taken by a time-based ANN for the prior example is presented in Figure 6. In this example, the maximum number of leaves to check is set to one; this returns the point $(0.330, 0.505)$ with distance 0.011 instead of the exact nearest-neighbor with distance 0.007. If the maximum number of leaves to check is set to two or more, we get the same result as the nearest-neighbors search Figure 5.

Distance-based approximations treat all potential neighbors within a fixed distance as ANNs. Unlike the time-based ANN method, this restriction does not necessarily improve the worst case performance of k-d trees. Instead, if the threshold is set too low an exact nearest-neighbor search will result. To ensure reduced run-time, some knowledge about appropriate distances is required.

In general, ties are not handled for values below the threshold. This lack of tie handling is potentially problematic if the order of evaluation of points in a leaf node are positively correlated with a variable of interest in the population.

The point returned as an ANN can be sensitive to the choice of threshold distance. In the case of the population in Figure 3, a threshold of distance 0.008 or higher will only explore the subset A_2 as in Figure 6. A threshold distance of 0.008 will return the same value as the time-based ANN with leaf check set at one. Likewise, a threshold set at 0.007 or lower will provide no reduction in leaf checks and return an exact nearest-neighbor. In an extreme case, a positive unbound threshold distance will return the first point found, with distance 0.285.

4. k-d Tree Acceleration

Integration of k-d tree-based nearest-neighbor searches in LPM2 is straight forward and allows for additional acceleration for simulation by preserving the tree structure between draws. For further performance gains, approximate nearest-neighbors k-d tree searches can be integrated in the same way. However, there is a trade-off to be made between the quality of the approximation and speed.

The LPM2 algorithm, Algorithm 1, can be modified to include k-d tree-based accelerated queries in two steps. First, a k-d tree must be built based on the points being sampled before line 1. Second, line 3 must be replaced with a k-d tree nearest-neighbor search or an approximate nearest-neighbor search. This modification will provide an exact replication of the results from the linear search implementation, provided that tied distances are handled in the same way. In k-d tree searches, the only additional burden is the cost of storing the tree in memory; fortunately, the storage requirements for the tree only grows linearly with respect to the number of points.

For simulations, multiple draws from the same sample do not require rebuilding the tree. This can make ANN searches considerably more desirable than using exact nearest-neighbor searches, since the approximate approach can have lower per-search cost and the tree building cost is amortized as the number of simulations increases.

ANN variants of LPM2 do make a trade-off between spatial balance and computational burden. By restricting the number of nodes to search, there is a chance that closer points may be ignored. Since there is a non-zero chance of exploring outside the node that the point being queried exists in, the joint inclusion probability of any two points remains non-zero. However, the joint inclusion probability can be substantially lowered for points that are spatially close but on opposite sides of a split that occurs early in tree formation.

5. Comparison of Algorithms

In this section, the performance and spatial balance of these algorithms are compared. This comparison is performed on simulated and real area frames. For simplicity, equal probability sampling is used for all comparisons. In the simulated cases, all populations are based on a bivariate uniform distribution of varying size distributed as $X_1 \sim \text{Uniform}(0, 1)$ and $X_2 \sim \text{Uniform}(0, 1)$, with X_1 independent of X_2, and a fixed sample size of 100. As an application to real data, the U.S. Census Bureau's 2010 block data are used. All calculations are performed using the R package *BalancedSampling* (see Grafström and Lisic, 2016).

To evaluate performance, a comparison of the log run-times is performed between the linear search and k-d tree implementations of the LPM2 algorithm, followed by a comparison between the run-times of the nearest-neighbor and ANN k-d tree implementations. Since the ANN variants of LPM2 do not necessarily preserve the nearest-neighbor, a short exploration of the effect of approximation on balance is performed. In this exploration, a comparison of spatial balance is performed between simple random sample (SRS), LPM2, and LPM2 approximations.

The log run-time for a k-d tree-based implementation of LPM2 is plotted against the linear run-time in Figure 7. This figure shows a dramatic reduction in run-time, in particular for a population size for 1,000,000 sampling units, the sample size is over eight orders of magnitude faster. Sampling a population of this size takes approximately three seconds using the k-d tree variant of LPM2, while the linear search implementation takes two and a half hours. Through extrapolation, a population of size 100,000,000 takes half an hour to perform with the k-d tree variant, while the linear search implementation finishes in forty-two days.

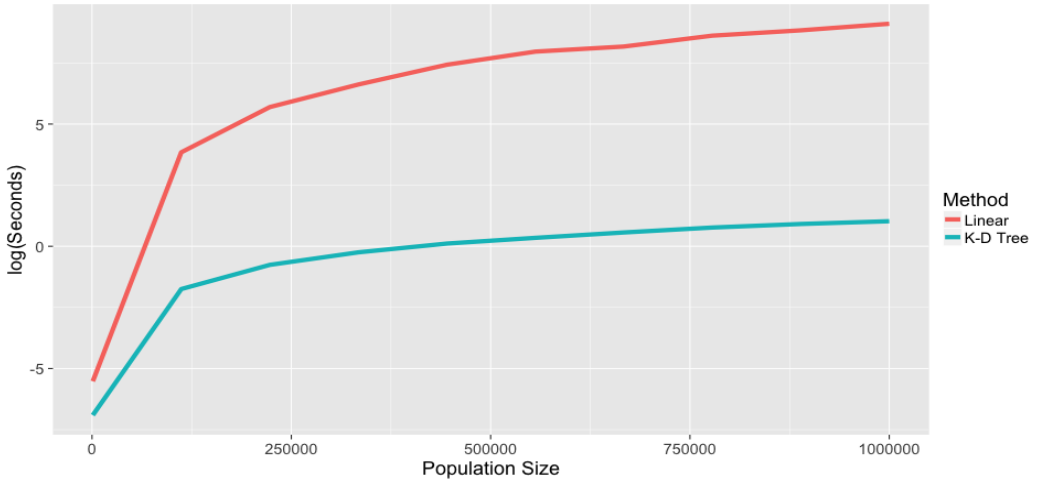

Figure 7: Log run-times to sample 100 sampling units using the linear search and k-d tree search based LPM2 methods from simulated populations with varying size distributed as $X_1 \sim \text{Uniform}(0, 1)$ and $X_2 \sim \text{Uniform}(0, 1)$.

As opposed to the extreme reduction in run-time seen with the prior comparison, the ANN speed reductions are considerably more modest (see Figure 8). In general, there is little difference between the run-times between the single leaf check and both distance approximations. This indicates a fairly early resolution to an acceptable ANN in these cases. The two leaf check ANN method on the other hand is generally on par with the exact nearest-neighbor implementation, indicating that for this population exact nearest-neighbor searches seldom exceeded two leaves.

A comparison of balance is also performed between the k-d tree-based nearest-neighbor and ANN methods. In this comparison, a few more levels are added to better understand the relationship between the approximation and balance. An SRS of the same population is provided as a reference. To estimate balance, 5,000 draws are performed for each sampling algorithm and parameterization. The standard error for the balance statistic B is approximately 0.01 in the worst case.

The result of this simulation indicated that the spatial methods are uniformly much more balanced than the SRS (see Table 1). Furthermore, the reduction in speed in the case of the more restrictive distance based ANNs comes at little expense in balance. In the case of time-based ANNs, the trade-off in balance is a bit more extreme, but unlike the more restrictive distance ANN, there is little reduction in computational burden relative to the exact nearest-neighbor LPM2.

The real data used for this research are the U.S. Census Bureau's census block data from the 2010 U.S. Census. These data are chosen due to their large size and the presence of spatial correlation between sampling units. The unit being sampled is a census block, the smallest areal unit available from the U.S. Census Bureau. The U.S. Census Bureau maintains census blocks throughout the United States, but in this research only a subset are used. This subset, which includes only the 48 contiguous states and the District of Columbia, is used as the population. Even with this reduction, the population includes 11,007,989 census blocks.

Geospatial polygons for these data are available for download from the U.S. Census Bureau with associated population and household size. For sampling, the centroids of the polygons are used to provide point data through longitude and latitude. Because census blocks with high population are likely to be close to other census blocks with high pop-

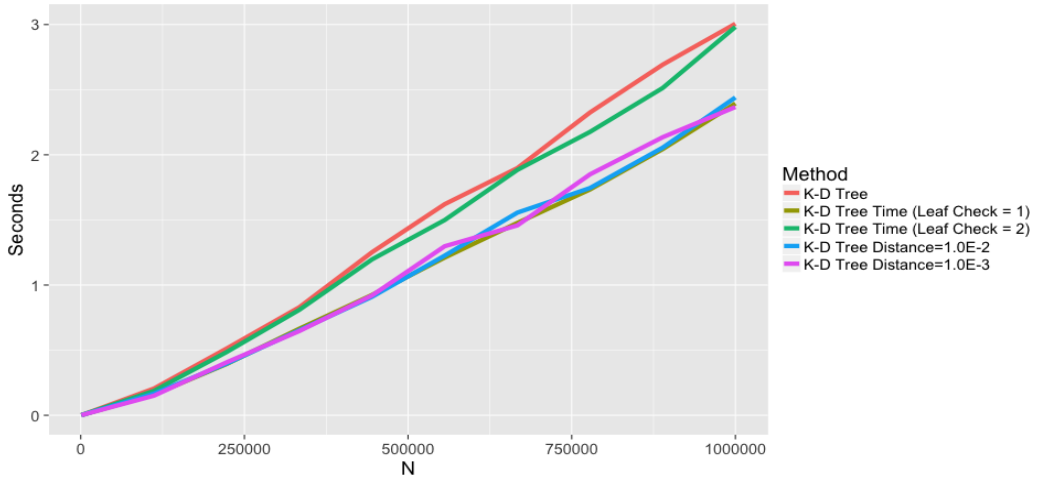

Figure 8: Run-times to sample 100 sample units using k-d tree-based LPM2 methods from simulated populations with varying size distributed as $X_1 \sim \text{Uniform}(0, 1)$ and $X_2 \sim \text{Uniform}(0, 1)$.

Table 1: A comparison of the balance statistic calculated from 5,000 draws per algorithm on a simulated population of 10,000 sampling units with sample size 100.

Algorithm	**Balance**
LPM2	0.07
SRS	0.32

k-d Tree Time	Leaf Check			
	1	2	3	4
	0.10	0.07	0.07	0.07

k-d Tree Dist.	Distance			
	1	0.1	0.01	0.001
	0.10	0.10	0.09	0.07

ulation, these data exhibit a high degree of spatial correlation between adjacent sampling units. In this experiment 5,000 samples of size 2,000 are drawn from the entire population. In each draw, a Horvitz-Thompson estimator of the U.S. Population for the 48 contiguous states is calculated. Owing to limitations of the software, each draw required rebuilding the k-d tree, significantly increasing the run-time.

The results of this simulation are provided in Table 2. In these results, variance relative to SRS is reduced by 9% for the approximate and 7% for the exact LPM2 sampling. Unfortunately, the sample variance for these results with 5,000 draws is sufficiently large to make the differences insignificant at a 5% confidence level. The difference in run-time is fairly insignificant with 45 seconds and 46 seconds for the time and distance based methods respectfully, and 58 seconds for the exact method.

Table 2: The mean of sample variances of Horvitz-Thompson estimates of the U.S. Population for the 48 contiguous states from a simulation of 5,000 draws.

Variance for Population Total

SRS	k-d Tree	k-d Tree Count Leaf Check = 1	k-d Tree Dist. Dist. = 0.01
3.655e+14	3.399e+14	3.370e+14	3.312e+14

6. Discussion

In this paper, a new k-d tree-based implementation of the LPM2 algorithm and two approximations to this algorithm are presented. Details and examples of the tree structure and nearest-neighbor searches are provided to allow for recreation of these methods. In application to synthetic data, these algorithms provide a reduction of run-times by eight orders of magnitude over the current LPM2 algorithm. These new implementations provide dramatic reduction in computational complexity and a means to exchange computational complexity for balance. The methods applied to U.S. Census census blocks show promising results, with variance reductions of 7-9% over a simple random sample. However, the simulation in this case is not of sufficient size to warrant a more definite conclusion.

The introduction of k-d tree-based nearest-neighbor searches in LPM2 allows for the application of spatially balanced sampling to frames consisting of up-to one billion sampling units and possibly larger frames. Given the growth of "Big Data," it is not inconceivable for spatial sampling frames of this magnitude to be in use in the near-future. Although not strictly in the domain of sample survey, this method does have immediate application in the sampling of high resolution geospatial imagery as a data reduction tool. In this capacity, this method has already been used to accelerate kernel density based classification of agricultural fields (see Lisic, 2015).

The acceleration of LPM2 does provide some immediately useful results; however, there are still a large number of questions regarding the statistical properties of the approximations. Future work will need to be done to better understand the relationship between the approximation, inclusion properties, and spatial balance.

Extension of this work to SPCM and to some extent GRTS should also be explored. In the case of SPCM, a simple replacement of the nearest-neighbor query with a k-d tree-based query should provide substantial reduction in computational cost. On the other hand, the use of k-d trees instead of quadtrees in GRTS should yield interesting results. In particular, k-d trees handle sparse data much better than quadtrees, provide better computational characteristics, and lower memory requirements.

References

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. and Wu, A. Y., 1998. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions." *Journal of the Association for Computing Machinery (JACM)* 45(6): 891–923.

Cressie, N. A. C., 1993. *Statistics for Spatial Data*. New York: Wiley.

Deville, J.-C. and Tille, Y., 1998. "Unequal probability sampling without replacement through a splitting method." *Biometrika* 85(1): 89–101.

Fattorini, L., 2006. "Applying the Horvitz-Thompson criterion in complex designs: a computer-intensive perspective for estimating inclusion probabilities." *Biometrika* 93(2): 269–278.

Grafström, A., 2012. "Spatially correlated Poisson sampling." *Journal of Statistical Planning and Inference* 142(1): 139–147.

Grafström, A. and Lisic, J., 2016. *BalancedSampling: Balanced and Spatially Balanced Sampling*. URL `https://CRAN.R-project.org/package=BalancedSampling`, r package version 1.5.2.

Grafström, A., Lundström, N. L. and Schelin, L., 2012. "Spatially Balanced Sampling through the Pivotal Method." *Biometrics* 68: 514–520.

Grafström, A., Saarela, S. and Ene, L. T., 2014. "Efficient sampling strategies for forest inventories by spreading the sample in auxiliary space." *Canadian Journal of Forest Research* 44(10): 1156–1164.

Kincaid, T. M. and Olsen, A. R., 2015. *spsurvey: Spatial Survey Design and Analysis*. URL `http://www.epa.gov/nheerl/arm/`, r package version 3.1.

Lang, D., Klaas, M. and de Freitas, N., 2005. "Empirical testing of fast kernel density estimation algorithms." *UBC Technical repor* 2.

Lisic, J., 2015. *Parcel Level Agricultural Land Cover Prediction*. Ph.D. dissertation, George Mason University.

Muja, M. and Lowe, D. G., 2014. "Scalable nearest neighbor algorithms for high dimensional data." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 36.

Stevens Jr, D. L. and Olsen, A. R., 2004. "Spatially balanced sampling of natural resources." *Journal of the American Statistical Association* 99(465): 262–278.

Sunter, A. B., 1977. "List sequential sampling with equal or unequal probabilities without replacement." *Journal of the Royal Statistical Society Series C (Applied Statistics)* 26(3): 261–268.

Tobler, W. R., 1970. "A computer movie simulating urban growth in the Detroit region." *Economic Geography* 46: 234–240.